



Semantics and Verification of Software

Winter Semester 2017/18

Lecture 9: Axiomatic Semantics of WHILE I (Hoare Logic)

Thomas Noll

Software Modeling and Verification Group

RWTH Aachen University

<http://moves.rwth-aachen.de/teaching/ws-1718/sv-sw/>

The Axiomatic Approach

The Axiomatic Approach I

Example 9.1

- Let $c \in \text{Cmd}$ be given by

`s := 0; n := 1; while $\neg(n > N)$ do s := s+n; n := n+1 end`

- How to show that, after termination of c ,

$$\sigma(s) = \sum_{k=1}^{\sigma(N)} k \quad ?$$

- “Running” c according to the operational semantics is insufficient: every change of $\sigma(N)$ requires a **new proof**
- Wanted: a more abstract, “**symbolic**” way of reasoning

The Axiomatic Approach

The Axiomatic Approach II

Example 9.1 (continued)

Obviously c satisfies the following **assertions** (after execution of the respective statement):

```
s := 0;  
{s = 0}  
n := 1;  
{s = 0 ∧ n = 1}  
while ¬(n > N) do s := s+n; n := n+1 end  
{s = ∑k=1N k ∧ n > N}
```

where, e.g., “ $s = 0$ ” means “ $\sigma(s) = 0$ in the current state $\sigma \in \Sigma$ ”

The Axiomatic Approach

The Axiomatic Approach III

How to prove the **validity** of assertions?

- Assertions following **assignments** are evident (“ $s = 0$ ”)
- Also, “ $n > N$ ” follows directly from the loop’s **execution condition**
- But how to obtain the final value of s ?
- Answer: after every loop iteration, the **invariant** $s = \sum_{k=1}^{n-1} k$ is satisfied
- Our proof system employs **partial correctness properties** of the form $\{A\} c \{B\}$ with assertions A, B and $c \in \text{Cmd}$
- Interpretation:

Validity of partial correctness property

$\{A\} c \{B\}$ is **valid** iff for all states $\sigma \in \Sigma$ which satisfy A :
if the execution of c in σ terminates in $\sigma' \in \Sigma$, then σ' satisfies B .

- “**Partial**” means that nothing is said about c if it fails to terminate
- In particular, $\{\text{true}\} \text{while true do skip end} \{\text{false}\}$ is a **valid** property

The Assertion Language

Syntax of Assertion Language I

Assertions = Boolean expressions + **logical variables**

- to memorize previous values of program variables
- to formulate more involved state properties

Syntactic categories:

Category	Domain	Meta variable(s)
Logical variables	<i>LVar</i>	<i>i</i>
Arithmetic expressions with logical variables	<i>LExp</i>	<i>a</i>
Assertions	<i>Assn</i>	<i>A, B, C</i>

The Assertion Language

Syntax of Assertion Language II

Definition 9.2 (Syntax of assertions)

The **syntax of *Assn*** is defined by the following context-free grammar:

$$\begin{aligned} a &::= z \mid x \mid i \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 * a_2 \in LExp \\ A &::= t \mid a_1 = a_2 \mid a_1 > a_2 \mid \neg A \mid A_1 \wedge A_2 \mid A_1 \vee A_2 \mid \forall i. A \in Assn \end{aligned}$$

- Thus: $AExp \subsetneq LExp$, $BExp \subsetneq Assn$
- The following (and other) **abbreviations** will be employed:

$$\begin{aligned} A_1 \Rightarrow A_2 &::= \neg A_1 \vee A_2 \\ \exists i. A &::= \neg(\forall i. \neg A) \\ a_1 \geq a_2 &::= a_1 > a_2 \vee a_1 = a_2 \\ &\vdots \end{aligned}$$

Semantics of Assertions

Semantics of $LExp$

The semantics now additionally depends on values of logical variables:

Definition 9.3 (Semantics of $LExp$)

An **interpretation** is an element of the set $Int := \{I \mid I : LVar \rightarrow \mathbb{Z}\}$. The **value of an arithmetic expressions with logical variables** is given by the functional

$$\mathcal{L}[\cdot] : LExp \rightarrow (Int \rightarrow (\Sigma \rightarrow \mathbb{Z}))$$

where

$$\begin{array}{ll} \mathcal{L}[z] l\sigma := z & \mathcal{L}[a_1 + a_2] l\sigma := \mathcal{L}[a_1] l\sigma + \mathcal{L}[a_2] l\sigma \\ \mathcal{L}[x] l\sigma := \sigma(x) & \mathcal{L}[a_1 - a_2] l\sigma := \mathcal{L}[a_1] l\sigma - \mathcal{L}[a_2] l\sigma \\ \mathcal{L}[i] l\sigma := I(i) & \mathcal{L}[a_1 * a_2] l\sigma := \mathcal{L}[a_1] l\sigma \cdot \mathcal{L}[a_2] l\sigma \end{array}$$

Definition 6.1 (denotational semantics of arithmetic expressions) implies:

Corollary 9.4

For every $a \in AExp$ (without logical variables), $I \in Int$, and $\sigma \in \Sigma$:

$$\mathcal{L}[a] l\sigma = \mathcal{U}[a] \sigma.$$

Semantics of Assertions

Semantics of Assertions I

- Formalized by a **satisfaction relation** of the form

$$\sigma \models A$$

(where $\sigma \in \Sigma$ and $A \in Assn$)

- Non-terminating computations captured by **undefined state** \perp :

$$\Sigma_{\perp} := \Sigma \cup \{\perp\}$$

- **Modification of interpretations** (in analogy to program states):

$$I[i \mapsto z](j) := \begin{cases} z & \text{if } j = i \\ I(j) & \text{otherwise} \end{cases}$$

Semantics of Assertions

Semantics of Assertions II

Reminder: $A ::= t \mid a_1 = a_2 \mid a_1 > a_2 \mid \neg A \mid A_1 \wedge A_2 \mid A_1 \vee A_2 \mid \forall i. A \in Assn$

Definition 9.5 (Semantics of assertions)

Let $A \in Assn$, $\sigma \in \Sigma_{\perp}$, and $I \in Int$. The relation “ σ satisfies A in I ” (notation: $\sigma \models^I A$) is inductively defined by:

$$\begin{array}{ll} \sigma \models^I \text{true} & \\ \sigma \models^I a_1 = a_2 & \text{if } \mathcal{L}[[a_1]]I\sigma = \mathcal{L}[[a_2]]I\sigma \\ \sigma \models^I a_1 > a_2 & \text{if } \mathcal{L}[[a_1]]I\sigma > \mathcal{L}[[a_2]]I\sigma \\ \sigma \models^I \neg A & \text{if not } \sigma \models^I A \\ \sigma \models^I A_1 \wedge A_2 & \text{if } \sigma \models^I A_1 \text{ and } \sigma \models^I A_2 \\ \sigma \models^I A_1 \vee A_2 & \text{if } \sigma \models^I A_1 \text{ or } \sigma \models^I A_2 \\ \sigma \models^I \forall i. A & \text{if } \sigma \models^{[i \rightarrow z]} A \text{ for every } z \in \mathbb{Z} \\ \perp \models^I A & \end{array}$$

Furthermore σ satisfies A ($\sigma \models A$) if $\sigma \models^I A$ for every interpretation $I \in Int$, and A is called **valid** ($\models A$) if $\sigma \models A$ for every state $\sigma \in \Sigma$.

Semantics of Assertions

Semantics of Assertions III

Example 9.6

The following assertion expresses that, in the current state $\sigma \in \Sigma$, $\sigma(y)$ is the greatest divisor of $\sigma(x)$:

$$(\exists i. i > 1 \wedge i * y = x) \wedge \forall j. \forall k. (j > 1 \wedge j * k = x \Rightarrow k \leq y)$$

In analogy to Corollary 9.4, Definition 6.2 (denotational semantics of Boolean expressions) yields:

Corollary 9.7

For every $b \in BExp$ (without logical variables), $l \in Int$, and $\sigma \in \Sigma$:

$$\sigma \models^l b \iff \mathfrak{B}[[b]]\sigma = \text{true}.$$

Semantics of Assertions

Semantics of Assertions IV

Definition 9.8 (Extension)

Let $A \in Assn$ and $I \in Int$. The **extension** of A with respect to I is given by

$$A' := \{\sigma \in \Sigma_{\perp} \mid \sigma \models' A\}.$$

Note that, for every $A \in Assn$ and $I \in Int$, $\perp \in A'$.

Example 9.9

For $A := (\exists i. i * i = x)$ and every $I \in Int$,

$$A' = \{\perp\} \cup \{\sigma \in \Sigma \mid \sigma(x) \in \{0, 1, 4, 9, \dots\}\}$$

Partial Correctness Properties

Partial Correctness Properties

Definition 9.10 (Partial correctness properties)

Let $A, B \in Assn$ and $c \in Cmd$.

- An expression of the form $\{A\} c \{B\}$ is called a **partial correctness property** with **precondition** A and **postcondition** B .
- Given $\sigma \in \Sigma_{\perp}$ and $I \in Int$, we let

$$\sigma \models' \{A\} c \{B\}$$

if $\sigma \models' A$ implies $\mathcal{C}[[c]]\sigma \models' B$ (or equivalently: $\sigma \in A' \Rightarrow \mathcal{C}[[c]]\sigma \in B'$).

- $\{A\} c \{B\}$ is called **valid in** I (notation: $\models' \{A\} c \{B\}$) if $\sigma \models' \{A\} c \{B\}$ for every $\sigma \in \Sigma_{\perp}$ (or equivalently: $\mathcal{C}[[c]]A' \subseteq B'$).
- $\{A\} c \{B\}$ is called **valid** (notation: $\models \{A\} c \{B\}$) if $\models' \{A\} c \{B\}$ for every $I \in Int$.

A Valid Partial Correctness Property

A Valid Partial Correctness Property

Example 9.11

- Let $x \in \text{Var}$ and $i \in \text{LVar}$. We have to show:

$$\models \{i \leq x\} x := x+1 \{i < x\}$$

- According to Definition 9.10, this is equivalent to

$$\sigma \models' \{i \leq x\} x := x+1 \{i < x\}$$

for every $\sigma \in \Sigma_{\perp}$ and $I \in \text{Int}$

- For $\sigma = \perp$ this is trivial. So let $\sigma \in \Sigma$:

$$\begin{aligned} & \sigma \models' (i \leq x) \\ \Rightarrow & \mathcal{L}[[i]]I\sigma \leq \mathcal{L}[[x]]I\sigma && \text{(Definition 9.5)} \\ \Rightarrow & I(i) \leq \sigma(x) && \text{(Definition 9.3)} \\ \Rightarrow & I(i) < \sigma(x) + 1 \\ & = (\mathcal{E}[[x := x+1]]\sigma)(x) \\ \Rightarrow & \mathcal{E}[[x := x+1]]\sigma \models' (i < x) \\ \Rightarrow & \text{claim} \end{aligned}$$

Proof Rules for Partial Correctness

Hoare Logic I

Goal: syntactic derivation of valid partial correctness properties. Here $A[x \mapsto a]$ denotes the syntactic replacement of every occurrence of x by a in A .



Tony Hoare (* 1934)

Definition 9.12 (Hoare Logic)

The **Hoare rules** are given by

$$\begin{array}{c} \text{(skip)} \frac{}{\{A\} \text{ skip } \{A\}} \\ \text{(seq)} \frac{\{A\} c_1 \{C\} \quad \{C\} c_2 \{B\}}{\{A\} c_1 ; c_2 \{B\}} \\ \text{(while)} \frac{\{A \wedge b\} c \{A\}}{\{A\} \text{ while } b \text{ do } c \text{ end } \{A \wedge \neg b\}} \\ \text{(asgn)} \frac{}{\{A[x \mapsto a]\} x := a \{A\}} \\ \text{(if)} \frac{\{A \wedge b\} c_1 \{B\} \quad \{A \wedge \neg b\} c_2 \{B\}}{\{A\} \text{ if } b \text{ then } c_1 \text{ else } c_2 \text{ end } \{B\}} \\ \text{(cons)} \frac{\models (A \Rightarrow A') \quad \{A'\} c \{B'\} \quad \models (B' \Rightarrow B)}{\{A\} c \{B\}} \end{array}$$

A partial correctness property is **provable** (notation: $\vdash \{A\} c \{B\}$) if it is derivable by the Hoare rules. In (while), A is called a **(loop) invariant**.

Proof Rules for Partial Correctness

Hoare Logic II

Example 9.13 (Factorial program)

Proof of $\{A\} y := 1; c \{B\}$ where

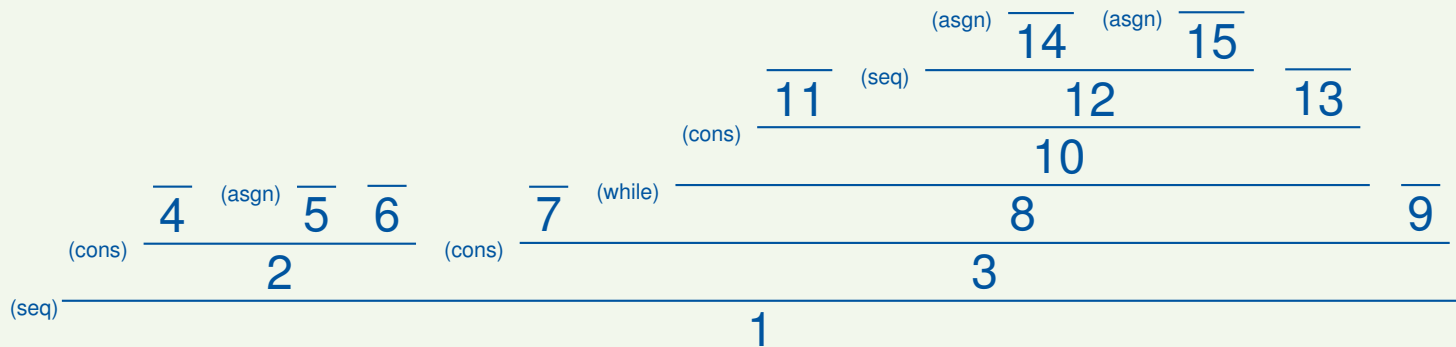
$$c := (\text{while } \neg(x=1) \text{ do } y := y*x; x := x-1 \text{ end})$$

$$A := (x > 0 \wedge x = i)$$

$$B := (y = i!)$$

(on the board)

Structure of the proof:



Proof Rules for Partial Correctness

Hoare Logic III

Example 9.13 (continued)

Here the respective propositions are given by (where $C := (x > 0 \wedge y * x! = i!)$):

1. $\{A\} y := 1; c \{B\}$
2. $\{A\} y := 1 \{C\}$
3. $\{C\} c \{B\}$
4. $\models (A \Rightarrow C[y \mapsto 1])$
5. $\{C[y \mapsto 1]\} y := 1 \{C\}$
6. $\models (C \Rightarrow C)$
7. $\models (C \Rightarrow C)$
8. $\{C\} c \{\neg(\neg(x = 1)) \wedge C\}$
9. $\models (\neg(\neg(x = 1)) \wedge C \Rightarrow B)$
10. $\{\neg(x = 1) \wedge C\} y := y*x; x := x-1 \{C\}$
11. $\models (\neg(x = 1) \wedge C \Rightarrow C[x \mapsto x-1, y \mapsto y*x])$
12. $\{C[x \mapsto x-1, y \mapsto y*x]\} y := y*x; x := x-1 \{C\}$
13. $\models (C \Rightarrow C)$
14. $\{C[x \mapsto x-1, y \mapsto y*x]\} y := y*x \{C[x \mapsto x-1]\}$
15. $\{C[x \mapsto x-1]\} x := x-1 \{C\}$